Metody obliczeniowe w nauce i technice - sprawozdanie 8

Łukasz Jezapkowicz

10.05.2020

1 Dany jest układ równań liniowych Ax=b.
Macierz A o wymiarze $n \times n$ jest określona wzorem:

$$\begin{bmatrix} 1 & \frac{1}{2} & 0 & ... & ... & ... & 0 \\ \frac{1}{2} & 2 & \frac{1}{3} & 0 & ... & ... & 0 \\ 0 & \frac{1}{3} & 2 & \frac{1}{4} & 0 & ... & 0 \\ ... & ... & ... & ... & ... & ... & ... \\ 0 & ... & ... & 0 & \frac{1}{n-1} & 2 & \frac{1}{n} \\ 0 & ... & ... & ... & 0 & \frac{1}{n} & 1 \end{bmatrix}$$

Przymij wektor x jako dowolną n-elementową permutację ze zbioru $\{-1, 0\}$ i oblicz wektor b (operując na wartościach wymiernych).
Metodą Jacobiego oraz metodą Czebyszewa rozwiąż układ równań liniowych $Ax = b$ (przyjmując jako niewiadomową wektor x).
W obu przypadkach oszacuj liczbę iteracji przyjmując test stopu:
$||x^{t+1} - x^t|| < \rho$
$\frac{1}{||b||}||Ax^{t+1} - b|| < \rho$

Program rozwiązujący ten problem zaimplementowałem w języku programowania $C++$.
Warto zacząć od funkcji generującej macierz A zgodną z macierzą podaną w zadaniu. Kod tworzący taką macierz widoczny jest poniżej:

```
/**
 * Function generating matrix A for given size n
 */
double* generate_matrix(int n) {
    double *A = new double[n*n];

    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            if (i == j) {
                if (i == 0 || i == n-1)
                    A[i*n+j] = 1.0;
                else
                    A[i*n+j] = 2.0;
            }
            else if (abs(i-j) == 1) {
                if (i < j)
                    A[i*n+j] = 1.0 / (i+2);
                else
                    A[i*n+j] = 1.0 / (j+2);
            }
            else
                A[i*n+j] = 0.0;
        }
    }

    return A;
}
```

Funkcja tworząca wektor x zgodnie z zadaniem widoczna jest poniżej:

```
/**
 * Function generating vector x for given size n
 */
double* generate_vector(int n) {
    double *x = new double[n];

    for (int i=0; i<n; i++) {
        int j = rand()%2;

        if (j == 0)
            x[i] = -1;
        else
            x[i] = 0;
    }

    return x;
}
```

Dla tak wygenerowanych danych funkcja obliczająca wektor wyrazów wolnych b widoczna jest poniżej:

```
/**
 * Function calculating vector b with given A and x
 */
double* find_b(double *A, double *x, int n) {
    double *b = new double[n];

    for (int i=0; i<n; i++) {
        double sum = 0.0;

        for (int j=0; j<n; j++)
            sum += 1.0*A[i*n+j]*x[j];

        b[i] = sum;
    }

    return b;
}
```

Kolejna funkcja rozwiązuje układ równań według metody Jacobiego, jej implementacja widoczna jest poniżej:

```
/**
 * Function calculating vector x in Ax=b by Jacobi method
 */
double* jacobi_method(double *A, double *b, int n, double error) {
    double *result_x = new double[n];
    for (int i=0; i<n; i++)
        result_x[i] = 0.0;

    double *L_U = new double[n*n];
    double *N = new double[n*n];
    double *M = new double[n*n];

    /* getting L_U,N matrixes */
    for (int i=0; i<n; i++)
        for(int j=0; j<n; j++) {
            L_U[i*n+j] = N[i*n+j] = 0;
            if (j == i)
                N[i*n+j] = pow(A[i*n+j], -1);
            else
                L_U[i*n+j] = A[i*n+j];
        }

    /* getting M matrix */
    for (int i=0; i<n; i++)
        for(int j=0; j<n; j++) {
            M[i*n+j] = -1 * N[i*n+i] * L_U[i*n+j];
        }
```

```
    /* iterations */
    int i = 0;
    while (true) {
        double *new_x = new double[n];
        for (int j=0; j<n; j++) {
            double mx = 0.0;
            double nb = 0.0;
            for (int k=0; k<n; k++) {
                mx += M[j*n+k]*result_x[k];
                nb += N[j*n+k]*b[k];
            }
            new_x[j] = mx+nb;
        }
        i++;
        double diff = 0.0;
        for (int j=0; j<n; j++) {
            diff += pow(result_x[j]-new_x[j],2);
        }

        if (sqrt(diff) < error)
            break;

        result_x = new_x;
    }

    cout << "It took " << i << " iterations to achieve result with maximum error = " << error << endl;
    delete[] L_U;
    delete[] N;
    return result_x;
}
```

Ostatnią ważną funkcją jest funkcja rozwiązująca układ według metody Czebyszewa:

```
/**
 * Function calculating vector x in Ax=b by Chebyshev method
 */
double* chebyshev_method(double *A, double *b, int n, double error) {
    double *result_x = new double[n];
    for (int i=0; i<n; i++)
        result_x[i] = 0.0;

    int i = 0;
    bool ready;
    double omega = 1.0;
    double rho = find_rho(A,b,n);
    while(true) {
        double add, diff = 0.0;
        for (int j=0; j<n; j++) {
            add = b[j];
            for (int k=0; k<n; k++) {
                add -= A[j*n+k]*result_x[k];
            }

            add = 1.0 * omega * add / A[j*n+j];
            diff += pow(add,2);
            result_x[j] += 1.0 * add;
            omega = 1.0 / (1 - 1.0 / 4.0 * omega * rho * rho);
        }
        i++;

        if (sqrt(diff) < error)
            break;
    }

    cout << "\nIt took " << i << " iterations to achieve result with maximum error = " << error << endl;
    return result_x;
}
```

gdzie $find\_rho$ jest dodatkową funkcją obliczającą promień spektralny.

Funkcja *main* wygląda następująco:

```
/**
 * program solving for the x in equation Ax = b
 */
int main(int argc, char *argv[]) {
    srand(time(NULL));

    /* getting the matrix A size */
    cout << "Give the A matrix size: ";
    int n;
    cin >> n;

    /* creating and generating matrix values */
    double *A = generate_matrix(n);
    //double A[] = {10,-1,2,-3,1,10,-1,2,2,3,20,-1,3,2,1,20};
    double *x = generate_vector(n);
    double *b = find_b(A,x,n);
    //double b[] = {0,5,-10,15};

    /* printing generated matrix and vectors */
    cout << "\nMatrix A:\n";
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            cout << left << setw(7) << setprecision(5) << A[i*n+j] << " ";
        }
        cout << endl;
    }
    cout << "\nVector x:\n";
    for (int i=0; i<n; i++)
        cout << x[i] << "\n";

    cout << "\nVector b:\n";
    for (int i=0; i<n; i++)
        cout << b[i] << "\n";
```

```
    /* solving for x in Ax=b by Jacobi method */
    double *result_x1 = jacobi_method(A,b,n,atof(argv[1]));

    cout << "\nJacobi method x:\n";
    double err = 0.0;
    for (int i=0; i<n; i++) {
        cout << result_x1[i] << "\n";
        err += 1.0 * pow(result_x1[i]-x[i],2);
    }
    cout << "\nTotal error = " << sqrt(err) << "\n";
    /* solving for x in Ax=b by Chebyshev method */
    double *result_x2 = chebyshev_method(A,b,n,atof(argv[1]));

    cout << "\nChebyshev method x:\n";
    err = 0.0;
    for (int i=0; i<n; i++) {
        cout << result_x2[i] << "\n";
        err += 1.0 * pow(result_x2[i]-x[i],2);
    }
    cout << "\nTotal error = " << sqrt(err) << "\n";

    // /* freeing memory */
    delete[] A;
    delete[] x;
    delete[] b;
    delete[] result_x1;
    delete[] result_x2;

    return 0;
}
```

Przykład działania dla pierwszego testu stopu ($||x^{t+1} - x^t|| < \rho$):

```
yyy@yyy-VirtualBox:~/Desktop/Lab8$ ./main 0.0000001
Give the A matrix size: 5

Matrix A:
1       0.5     0       0       0
0.5     2       0.33333 0       0
0       0.33333 2       0.25    0
0       0       0.25    2       0.2
0       0       0       0.2     1

Vector x:
0
-1
-1
-1
-1

Vector b:
-0.5
-2.3333
-2.5833
-2.45
-1.2
It took 20 iterations to achieve result with maximum error = 1e-07

Jacobi method x:
-3.1827e-08
-1
-1
-1
-1

Total error = 3.5909e-08

It took 8 iterations to achieve result with maximum error = 1e-07

Chebyshev method x:
1.0646e-08
-1
-1
-1
-1

Total error = 1.1192e-08
```

Jak widać metoda Czebyszewa okazała się tu 2.5 razy szybsza.

Przykład działania dla drugiego testu stopu ($\frac{1}{||b||}||Ax^{t+1} - b|| < \rho$):

```
yyy@yyy-VirtualBox:~/Desktop/Lab8$ ./main 0.0000001
Give the A matrix size: 5

Matrix A:
1       0.5     0       0       0
0.5     2       0.33333 0       0
0       0.33333 2       0.25    0
0       0       0.25    2       0.2
0       0       0       0.2     1

Vector x:
0
-1
-1
-1
-1

Vector b:
-0.5
-2.3333
-2.5833
-2.45
-1.2
It took 17 iterations to achieve result with maximum error = 1e-07

Jacobi method x:
2.281e-07
-1
-1
-1
-1

Total error = 4.7988e-07

It took 6 iterations to achieve result with maximum error = 1e-07

Chebyshev method x:
2.0395e-07
-1
-1
-1
-1

Total error = 2.4124e-07
```

Jak widać metoda Czebyszewa okazała się tu 2.8(3) razy szybsza.

Wniosek: Metody iteracyjne rozwiązujące układy równań są o wiele szybsze od ich bezpośrednich odpowiedników. Ich implementacja nie jest nadto skomplikowana, lecz nie jest też trywialna. Jak pokazało doświadczenie metoda Czebyszewa jest o wiele szybsza niż podstawowa metoda jaką jest metoda Jacobiego. Metoda Czebyszewa jest również w miejscu co jest kolejną jej zaletą. Warto więc używać bardziej skomplikowanych metod iteracyjnych, ponieważ mają one dużo plusów w stosunku do prostszych metod.

2 Dowieść, że proces iteracji dla układu równań:

$$10x_1 - x_2 + 2x_3 - 3x_4 = 0$$
$$x_1 + 10x_2 - x_3 + 2x_4 = 5$$
$$2x_1 + 3x_2 + 20x_3 - x_4 = -10$$
$$3x_1 + 2x_2 + x_3 + 20x_4 = 15$$

jest zbieżny. Ile iteracji należy wykonać, żeby znaleźć pierwiastki układu z dokładnością do 10^{-3},10^{-4},10^{-5}**?**

Rozwiązanie napisane ręcznie widoczne jest na kolejnej stronie.

Zad. 2.

Układ równań macierzowo:

$$A = \begin{bmatrix} 10 & -1 & 2 & -3 \\ 1 & 10 & -1 & 2 \\ 2 & 3 & 20 & -1 \\ 3 & 2 & 1 & 20 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \\ -10 \\ 15 \end{bmatrix}$$

Poszukuję $\rho(M)$. M poszukam dla metody Jacobiego.

$$M = I - D^{-1}A$$

$$D = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 20 \end{bmatrix} \Rightarrow D^{-1} = \begin{bmatrix} \frac{1}{10} & 0 & 0 & 0 \\ 0 & \frac{1}{10} & 0 & 0 \\ 0 & 0 & \frac{1}{20} & 0 \\ 0 & 0 & 0 & \frac{1}{20} \end{bmatrix}$$

$$M = I - D^{-1} \cdot A = \begin{bmatrix} 0 & \frac{1}{10} & -\frac{1}{5} & \frac{3}{10} \\ -\frac{1}{10} & 0 & \frac{1}{10} & -\frac{1}{5} \\ -\frac{1}{10} & -\frac{3}{20} & 0 & \frac{1}{20} \\ -\frac{3}{20} & -\frac{1}{10} & -\frac{1}{20} & 0 \end{bmatrix}$$

$$\det(M - \lambda I) = \lambda^4 + 0.0325\lambda^2 - 0.003\lambda + 0.0004$$

$$\lambda_1 = -0.060385 + 0.179768\,i$$

$$\lambda_2 = -0.060385 - 0.179768\,i$$

$$\lambda_3 = 0.060385 + 0.179768\,i$$

$$\lambda_4 = 0.060385 - 0.179768\,i$$

$$\rho(M) = \max_i |\lambda_i| = |\lambda_1| \approx 0.189642 < 1 \rightarrow \text{układ jest iteracyjnie zbieżny}$$

Ilość iteracji w celu osiągnięcia dokładności 10^{-p} (zgodnie z wykładem):

$$t^* = \frac{p}{R} \quad \text{gdzie} \quad R = -\log_{10}(\rho(M)) \approx 0.722065$$

Dla $p = 3$

$t^* \approx 4.15\ldots \Rightarrow$ 5 iteracji

Dla $p = 4$

$t^* \approx 5.53\ldots \Rightarrow$ 6 iteracji

Dla $p = 5$

$t^* \approx 6.92\ldots \Rightarrow$ 7 iteracji

Wniosek: Posługując się konkretną metodą (np. Jacobiego) można łatwo sprawdzić czy proces iteracji dla danego układu równań jest zbieżny. W tym celu musimy obliczyć promień spektralny macierzy iteracyjnej M. Przy jego pomocy możemy łatwo obliczyć ilość iteracji potrzebną do osiągnięcia zamierzonej dokładności rozwiązania.